A10

Hardware Performance Monitoring Tools and APIs

Luc Smolders

IBM SYSTEM p, AIX 5L and LINUX TECHNICAL UNIVERSITY
Sept 11 - 15, 2006

Las Vegas, NV

© IBM Corporation 2006

© IBM Corporation 2006

Agenda

- Hardware Performance Monitoring introduction
- PMAPI
  - introduction
  - common rules
  - security
  - simple examples
  - TL5 update
  - recent processors support
  - Power4 and Power5 cpi stacks
- pmlist utility
- libhpm
  - introduction
  - simple examples
- Tools
  - hpmcount
  - hpmstat
  - event-based profiling
    - tprof -E

© IBM Corporation 2006

Hardware Performance Monitoring - introduction

- PowerPC processors have 2 to 8 programmable counters

- Many different types of events (counts or duration) can be monitored, e.g.
  - hits, miss and latencies in various cache hierarchy levels
  - instruction types (e.g. floating point loads, FMAs, sync, ...)
  - completion delays
  - unit usage
  - queue occupancy

- Recent processors also support profiling
  - counter overflows can be made to generate interrupts
  - two registers (SIA and SDA) capture instruction and data address
    - automatically frozen on counter overflowing

- AIX support is in fileset bos.pmapi, which includes
  - pmsvcs kernel extension
  - libpmapi and libhpm libraries
  - pmlist, hpmcount, and hpmstat tools

© IBM Corporation 2006

PMAPI - introduction

- Kept as simple as possible
  - table driven
    - hiding Power4/PowerPC970/Power5 event selection complexity from users
  - to be able to tolerate processor differences
    - code is totally processor agnostic
    - currently supporting 12 types of processors

- Maintains 64bit software counters
  - virtualized mode
    - supports both kernel threads and pthreads
    - supports threads grouping
      - threads with common ancestor
      - automatically accumulates counts for group
    - counters and groups automatically propagated on thread and process creations
  - system-wide mode
    - automatic overflow accumulation
    - support process tree mode
      - count a family of process with common ancestor

© IBM Corporation 2006

PMAPI - basic interfaces

- There are a total of 5 basic set of APIs
  - system level API calls: to monitor whole machine or process tree activity
  - first party thread API calls : to monitor current kernel thread activity
  - first party thread group API calls : to monitor group of threads in current process
  - third party thread API calls : to monitor a thread in a debuggee process
  - third party thread group API calls : to monitor group of threads in debuggee

- Each set includes 7 basic calls
  - **pm_set_program** : to program Performance Monitor with list of events and mode
  - **pm_get_program** : returns mode and list of events being counted
  - **pm_delete_program** : undoes pm_set_program
  - **pm_start** : starts the counting
  - **pm_stop** : stops the counting
  - **pm_get_data** : collects 64bit software counters, one per hardware counter
  - **pm_reset_data** : resets counts

- Actual calls are variations from system level API names using suffix
  - **_mythread** and **_mygoup** for first party calls (ex: **pm_get_data_mygroup)**
  - **_pthread, _thread** and **_group** for third party calls (ex: **pm_start_thread)**

© IBM Corporation 2006

PMAPI - common rules

- **pm_initialize** must be called before any other API call can be made
  - returns list of events for each available hardware counter
    - identifier : to be used with **pm_set_program** and **pm_get_program** calls
    - short name : mnemonic name for easy searching (see cpi example)
    - long name : full name
    - description : full description of event (from hardware documentation)
    - event status and charactericties
      - testing status: verified, caveat or unverified(use at your own risks)
      - charactericties: thresholdable, group only, shared, ...
  - on POWER4 and later, also returns available groups of events
  - processor characteristics
    - name, number of counters, threshold multipliers, features supported
- Input is a mask for event testing status bits and optional processor name
  - only events with requested status will be returned
  - only event returned can be used in subsequent calls
  - can retrieve tables for other processors
- No reprogramming is allowed
  - call to **pm_delete_program_*** must be made before a new call to **pm_set_program_*** can be made

© IBM Corporation 2006

PMAPI - security

- System level APIs only available to super user
  - except when process tree option is used
  - locking mechanism prevents more than one system level session at a time
    - including profiling session
  - locking also applies between system level API and any thread level API call
    - system level API would return incorrect results if thread level counting was on

- Third party call rules
  - target thread or group of thread must be a debuggee process of caller
    - debuggee must either be ptraced by caller
    - or caller must have write access to its control file in /proc
  - debuggee must be stopped
  - same security as ptrace/debugger or /proc

© IBM Corporation 2006

PMAPI - simple example

```
#include <pmapi.h>

main()
{
    pm_info2_t pminfo;
    pm_prog_t prog;
    pm_data_t data;
    pm_groups_info_t pmginfo;
    int filter = PM_VERIFIED|PM_GET_GROUPS; /* only verified events/groups */

    pm_initialize(filter, &pminfo, &pmginfo, PM_CURRENT)

    prog.mode.w           = 0;  /* start with clean mode */
    prog.mode.b.user      = 1;  /* count only user mode */
    prog.mode.b.is_group  = 1;  /* using group counting mode */

    for (i = 0; i < pminfo.maxpmcs; i++)
            prog.events[i] = COUNT_NOTHING;

    prog.events[0] = 1;  /* count event 1 in first counter or group 1 */
    prog.events[1] = 2;  /* count event 2 in second counter (ignored) */

    pm_program_mythread(&prog);
    pm_start_mythread();

(1)  ... workload to measure ....

    pm_stop_mythread();
    pm_get_data_mythread(&data);
 }
```

© IBM Corporation 2006

PMAPI - simple multithreaded example

```
pm_data_t data2;

void *
doit(void *)
{

(1)    pm_start_mythread();

       ... workload to measure ....

       pm_stop_mythread();
       pm_get_data_mythread(&data2);
}
```

- Auxiliary thread inherited PM programming from main thread
- Counting starts at (1) and (2) for the main and auxiliary threads respectively because the initial counting state was off and it was inherited by the auxiliary thread from its creator.

```
main()
{
       pthread_t        threadid;
       pthread_attr_t attr;
       pthread_addr_t status;

       ... same initialization as in previous example ...

       pm_program_mythread(&prog);

       pthread_attr_init(&attr);
       pthread_create(&threadid, &attr, doit, NULL);

(2)    pm_start_mythread();

       ... usefull work ....

       pm_stop_mythread();
       pm_get_data_mythread(&data);

       ... print main thread results (data )...

       pthread_join(threadid, &status);

       ... print auxiliary thread results (data2) ...
}
```

© IBM Corporation 2006

PMAPI - thread counting group example

```
main()
 {
     ... same initialization and doit code as in previous example ...

     pm_program_mygroup(&prog); /* create counting group */
(1)  pm_start_mygroup()

     pthread_create(&threadid, &attr, doit, NULL)

(2)  pm_start_mythread();

     ... usefull work ....

     pm_stop_mythread();
     pm_get_data_mythread(&data)

     ... print main thread results ...

     pthread_join(threadid, &status);

     ... print auxiliary thread results ...

     pm_get_data_mygroup(&data)

     ... print group results ...
 }
```

- The call in (2) is necessary because the call in (1) only turns on counting for the group, not for the individual threads in it. At the end, the group results are the sum of both thread results.

© IBM Corporation 2006

PMAPI - debugger example

- To look at the PM data while the first sample program is executing
  - from a debugger at breakpoint (1)

```
      pm_initialize(filter, &pminfo, &pmginfo, PM_CURRENT);

(2)   pm_get_program_thread(pid, tid, &prog);

      ... display PM information ...

(3)   pm_get_data_thread(pid, tid);

      ... display PM data ...

      pm_delete_program_thread(pid, tid);
      prog.events[0] = 2; /* change counter 1 to count group number 2 */
      pm_set_program_thread(pid, tid, &prog);
```

  - continue program
- The scenario above would work as well if the program being executed under the debugger didn't have any embedded PM API calls.
- The only difference would be that the calls at (2) and (3) would fail

© IBM Corporation 2006

PMAPI - TL 5 update

- New set of APIs reporting time
  - **pm_tstart*** and **pm_tstop***
    - return timestamps(time base values) when counting started or stopped
  - **pm_get_tdata*** interfaces to measure counting intervals
    - return timestamps(time base values) when hardware counters were last read
    - can be used in combination with **pm_get_tstart***
  - **pm_get_Tdata***
    - report measurement interval in TB, PURR and SPURR units, e.g.

```
typedef struct {
    timebasestruct_t accu_timebase;  /* accumulated time base   */
    timebasestruct_t accu_purr;      /* accumulated PURR time   */
    timebasestruct_t accu_spurr;     /* accumulated SPURR time */
 } pm_accu_time_t;

pm_get_Tdata(pm_data_t *data, pm_accu_time_t *times);
```

- Counter multiplexing
  - ability to count more events than number of physical counters
  - supported by libpmapi, libhpm, hpmcount and hpmstat
    - new set of **pm_*_mx** interfaces
    - expanded command line syntax for hpmcount and hpmstat to support multiple event sets
    - expanded syntax for libhpm/hpmcount/hpmstat environment variables to support multiple event sets
- Dynamic Reconfiguration support

© IBM Corporation 2006

PMAPI - counter multiplexing

- New data structures

```
typedef int pm_events_prog_t[MAX_COUNTERS];
typedef struct {                              /* structure for PM programing        */
  pm_mode_t          mode;                    /* mode of operation                  */
  int                slice_duration;          /* duration of each time slice in ms  */
  int                nb_events_prog;          /* number of events_set               */
  pm_events_prog_t *events_set;               /* list of counted events             */
} pm_prog_mx_t;

typedef struct {
  timebasestruct_t accu_time;                 /* accumulated time                   */
  timebasestruct_t accu_purr;                 /* accumulated PURR time              */
  timebasestruct_t accu_spurr;                /* accumulated SPURR time             */
  long long          accu_data[MAX_COUNTERS]; /* accumulated data                   */
} pm_accu_mx_t;

typedef struct {                              /* structure for PM data              */
  pm_ginfo_t    ginfo;                        /* group information                  */
  int           nb_accu_mx;                   /* number of accu_set                 */
  int           nb_mx_round;                  /* number of loops on all the event sets */
  pm_accu_mx_t *accu_set;                     /* accumulated data                   */
} pm_data_mx_t;
```

- Example of new interfaces

```
int    pm_set_program_mx(pm_prog_mx_t *prog)     [compares to pm_set_program(pm_prog_t *prog)]
int    pm_get_program_mx(pm_prog_mx_t *prog)     [compares to pm_get_program(pm_prog_t *prog)]
int    pm_get_data_mx(pm_data_mx_t *data)        [compares to pm_get_data(pm_data_t *data)]
```

© IBM Corporation 2006

PMAPI - Dynamic Reconfiguration support

- Processor additions and deletion now supported
  - includes turning SMT on or off

- Impact to per-cpu interfaces
  - pm_get_data_cpu, pm_get_tdata_cpu and the new pm_get_Tdata_cpu and pm_get_data_cpu_mx interfaces
    - cpuids are always contiguous (0 to __systemcfg.**ncpus**)
    - may not always represent the same logical processors
    - DR operations renumber cpus
    - partial results for deleted cpus are lost
  - new pm_get_data_lcpu, pm_get_tdata_lcpu, pm_get_Tdata_lcpu, and pm_get_data_lcpu_mx interfaces
    - cpuids are not always contiguous (0 to __systemcfg.**max_ncpus**)
    - always represent the same logical processor
    - DR operations create or fill holes in lcpuids
    - partial results for deleted cpus can be retrieved

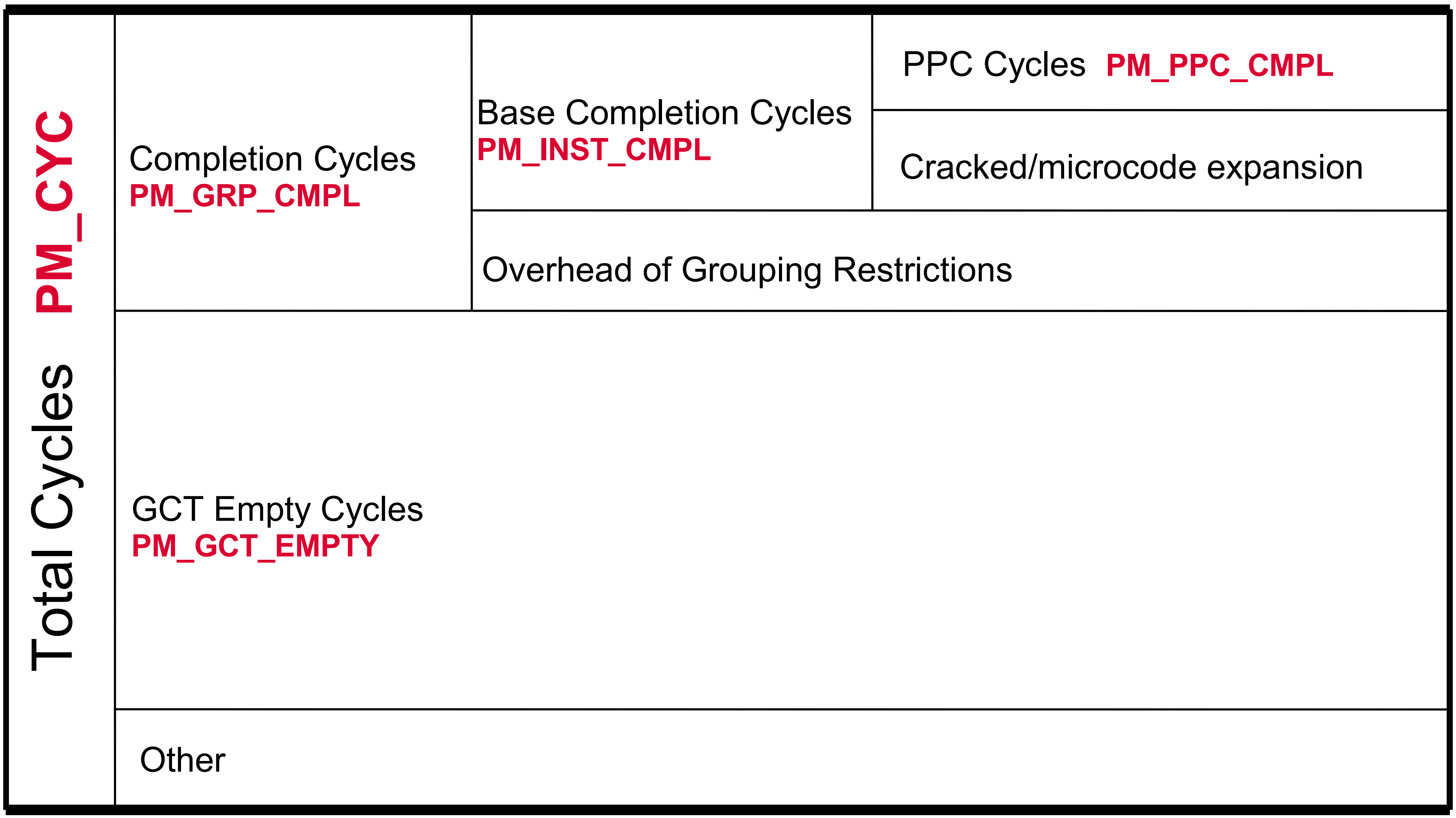
© IBM Corporation 2006

PMAPI - recent processors support

- PowerPC 970
  - 8 programmable counters, 470 events, currently 50 groups
  - very similar events and groups than Power4
    - new VMX events exists but are not in current tables
- Power5
  - 6 counters, 4 programmables
    - pmc5 always counts PM_INST_CMPL (instructions completed)
    - pmc6 always counts PM_RUN_CYC (run/busy cycles)
  - 470 events, 144 groups
- Power5+
  - pmc5 now counting PM_RUN_INST_CMPL (run/busy instructions completed)
  - pmc6 still counting PM_RUN_CYC
  - 183 groups
    - very similar to Power5 in content but numbers are not compatible

© IBM Corporation 2006

PMAPI - POWER4 cpi stack



© IBM Corporation 2006

PMAPI - POWER5 cpi stack

Total Cycles PM_CYC	Completion Cycles PM_GRP_CMPL	Base Completion Cycles PM_INST_CMPL	PPC Cycles PM_PPC_CMPL
			Cracked/microcode expansion
		Overhead of Grouping Restrictions	
	GCT Empty Cycles PM_GCT_EMPTY	Icache Miss PM_GCT_NOSLOT_IC_MISS	
		Branch Mispredict PM_GCT_NOSLOT_BR_MPRED	
		SRQ Full PM_GCT_NOSLOT_SRQ_FULL	
		Other	
	Stall by LSU PM_CMPLU_STALL_LSU	Reject PM_CMPLU_STALL_REJECT	Translation PM_CMPLU_STALL_ERAT_MISS
			Other
		Dcache Miss PM_CMPLU_STALL_DCACHE_MISS	
		Basic Latency, Flush overhead	
	Stall by FXU PM_CMPLU_STALL_FXU	Long Latency Ops PM_CMPLU_STALL_DIV	
		Other	
	Stall by FPU PM_CMPLU_STALL_FPU	Long Latency Ops PM_CMPLU_STALL_FDIV	
		Other	
	Other		

© IBM Corporation 2006

CPI analysis on POWER5

- Reference articles
  - tools

    http://www-128.ibm.com/developerworks/power/library/pa-cpipower1

  - cpi breakdown model

    http://www-128.ibm.com/developerworks/power/library/pa-cpipower2

© IBM Corporation 2006

PM tools - pmlist command

- pmlist
  - utility to display and search processors event, group and derived metrics tables
  - currently supports test and comma separated formats
- Usage

```
usage: pmlist -h
       pmlist -l [ -o t | c ]
       pmlist -s | -e <short|select> | -c counter[,event] | -g group | -S set | -D DerivedMetric
                [-p procname] [-s] [-d] [-o t|c] [-f filter]
where:
   -h                    this help screen
   -l                    lists all supported processor types
   -s                    displays processor information summary
   -e short|select       lists all events with this short name or select event value
   -c -1                 lists all events for all counters
   -c counter            lists all events for the specified counter
   -c counter,event      lists the specified event for the specified counter
   -D -1                 lists all the derived metrics
   -D DerivedMetric      lists detailed information for the specified derived metric
   -g -1                 lists all the event groups
   -g group              lists the specified event group
   -S -1                 lists all the event sets
   -S set                lists the specified event set
   -p procname           specifies the processor for which information will be listed
   -d                    displays event detailed description
   -o format             specifies the output format:
                            t is for text (default)
                            c is for comma separated values
   -f v,u,c              specifies the event filters (default is v,u,c).
                         these represent the testing status of an event:
                            v is for verified
                            u is for unverified
                            c is for caveat
```

© IBM Corporation 2006

PM tools - pmlist examples

- pmlist -l

```
Processors supported (specify with -p)
====================
PowerPC604
PowerPC604e
RS64-II
POWER3
RS64-III
POWER3-II
POWER4
MPC7450
POWER4-II
POWER5
PowerPC970
POWER5-II
```

- pmlist -p PowerPC970 -c 1,4 -d

```
Event #  Status  group Threshold share  Short Name  Long Name Description
=== Counter 1
#4,v,g,n,n,PM_DATA_TABLEWALK_CYC,Cycles doing data tablewalks
   This signal is asserted every cycle when a tablewalk is active. While a tablewalk is active any request attempting
  to access the TLB will be rejected and retried.
```

- pmlist -p POWER4 -e PM_INST_CMPL

```
POWER4: information about PM_INST_CMPL event

Event#,Status,Grouped,Threshold,Shared,SelectEvent,ShortName,LongName
=== Pmc 1
 86,c,g,n,n,8001,PM_INST_CMPL,Instructions completed
=== Pmc 2
=== Pmc 3
=== Pmc 4
 77,c,g,n,n,8001,PM_INST_CMPL,Instructions completed
=== Pmc 5
=== Pmc 6
 86,c,g,n,n,8001,PM_INST_CMPL,Instructions completed
=== Pmc 7
 78,c,g,n,n,8001,PM_INST_CMPL,Instructions completed
=== Pmc 8
 81,c,g,n,n,8001,PM_INST_CMPL,Instructions completed
```

© IBM Corporation 2006

PM tools - pmlist examples *(cont)*

- pmlist -p POWER5 -g -1

```
# of groups: 144.

Group #0: pm_utilization
Group name: CPI and utilization data
Group description: CPI and utilization data
Group status: Verified
Group members:
Counter  1, event 190: PM_RUN_CYC  : Run cycles
Counter  2, event 71: PM_INST_CMPL  : Instructions completed
Counter  3, event 56: PM_INST_DISP  : Instructions dispatched
Counter  4, event 12: PM_CYC [shared] : Processor cycles
Counter  5, event  0: PM_INST_CMPL  : Instructions completed
Counter  6, event  0: PM_RUN_CYC  : Run cycles

Group #1: pm_completion
Group name: Completion and cycle counts
Group description: Completion and cycle counts
Group status: Verified
Group members:
Counter  1, event  2: PM_1PLUS_PPC_CMPL  : One or more PPC instruction completed
Counter  2, event 195: PM_GCT_EMPTY_CYC [shared] : Cycles GCT empty
Counter  3, event 49: PM_GRP_CMPL  : Group completed
Counter  4, event 12: PM_CYC [shared] : Processor cycles
Counter  5, event  0: PM_INST_CMPL  : Instructions completed
Counter  6, event  0: PM_RUN_CYC  : Run cycles

...

Group #143: pm_hpmcount4
Group name: HPM group for set 7
Group description: HPM group for set 7
Group status: Verified
Group members:
Counter  1, event 210: PM_TLB_MISS  : TLB misses
Counter  2, event 15: PM_CYC [shared] : Processor cycles
Counter  3, event 165: PM_ST_REF_L1  : L1 D cache store references
Counter  4, event 106: PM_LD_REF_L1  : L1 D cache load references
Counter  5, event  0: PM_INST_CMPL  : Instructions completed
Counter  6, event  0: PM_RUN_CYC  : Run cycles
```

© IBM Corporation 2006

HPM library - introduction

- Higher-level (simpler) instrumentation library for Fortran, C, and C++
  - 4 interfaces: **hpmInit()**, **hpmStart()**, **hpmStop()** and **hpmTerminate()**
  - parametrization completely done via environment variables
    - no complicated set of arguments to pass to APIs
    - no need to recompile to count different events
  - **hpmTerminate()** prints results to file
- Supports
  - MPI, OpenMP, and pthreads
  - multiple instrumentation points
  - nested instrumentation
  - multiple calls to an instrumented point
- For the total execution of the instrumented program, provides
  - resource usage statistics
- For each instrumented section of code, provides
  - total count and duration (wall clock time)
  - hardware performance counters information
  - derived metrics

© IBM Corporation 2006

HPM library - derived metrics

- **Hardware events**
  - cycles
  - Instructions
  - Floating point instructions
  - Integer instructions
  - Load/stores
  - Cache misses
  - TLB misses
  - Branch taken/not taken
  - Branch mispredictions

- **Derived metrics**
  - IPC – instructions per cycle
  - Floating point rate (Mflip/s)
  - FP computation intensity(flip per load/store)
  - Instructions per load/store
  - Load/stores per cache miss
  - Cache hit rate
  - Loads per load miss
  - Stores per store miss
  - Loads per TLB miss
  - Branch mispredicted %

- Derived metrics are automatically calculated when **hpmTerminate()** is called

© IBM Corporation 2006

HPM library - derived metrics examples

- pmlist -p PowerPC604e -D -1

```
Derived metrics supported:
        PMD_PROC_TIME                  Processing time
        PMD_UTI_RATE                   Utilization rate
        PMD_INST_PER_CYC               Instructions per cycle
        PMD_MIPS                       MIPS
        PMD_PRC_INST_DISP_CMPL         % Instructions dispatched that completed
        PMD_LD_ST                      Total load and store operations
        PMD_INST_PER_LD_ST             Instructions per load/store
        PMD_INST_PER_IC_MISS           Instructions per I Cache Miss
        PMD_PRC_LSU_IDLE               % Cycles LSU is idle
        PMD_SNOOP_RATE                 Snoop hit rate
        PMD_HW_FP_PER_CYC              HW floating point instructions per Cycle
        PMD_HW_FP_PER_UTIME            HW floating point instructions / user time
        PMD_HW_FP_RATE                 HW floating point rate
        PMD_FX                         Total Fixed point operations
        PMD_FX_PER_CYC                 Fixed point operations per Cycle
        PMD_TLB_EST_LAT                Estimated latency from TLB miss
        PMD_MBR_PRC                    Branches mispredicated percentage
```

- pmlist -D PMD_MIPS

```
Derived Metric: PMD_MIPS (MIPS)
Formula       : (0.000001 * PM_INST_CMPL) / total_time
Description   :

Sets          : 1,2,3,4,5,6,7,8
```

- pmlist -D PMD_MBR_PRC

```
Derived Metric: PMD_MBR_PRC (Branches mispredicated percentage)
Formula       : (PM_BR_MPRED * 100.) / PM_BR_DISP
Description   :

Sets          : 8
```

© IBM Corporation 2006

HPM library - simple example

```
do_work()
{
     pid_t p;
     float f1 = 9.7641, f2 = 2.441, f3 = 0.0;
     f3 = f1 / f2;

     printf("f3=%f\n", f3);

     p = fork();
     if (p == -1) {
       perror("fork error");
       exit(1);
     }

     if (p == 0)
       execl("/usr/bin/sh", "sh", "-c", "ls -R / 2>&1 >/dev/null", 0);
     else
       waitpid(p, &status, WUNTRACED | WCONTINUED);
}

main(int argc, char **argv)
{
     int taskID = 999;

     hpmInit(taskID, "my_program");
     hpmStart(1, "outer call");
     do_work();
     hpmStart(2, "inner call");
     do_work();
     hpmStop(2);
     hpmStop(1);
     hpmTerminate(taskID);
}
```

© IBM Corporation 2006

HPM library - sample program output

```
Total execution time of instrumented code (wall time): 2.204872 seconds

 ########  Resource Usage Statistics  ########

 Total amount of time in user mode              : 0.007864 seconds
 Total amount of time in system mode            : 0.003551 seconds
 Maximum resident set size                      : 864 Kbytes
 Average shared memory use in text segment      : 0 Kbytes*sec
 Average unshared memory use in data segment    : 0 Kbytes*sec
 Number of page faults without I/O activity     : 310
 Number of page faults with I/O activity        : 0
 Number of times process was swapped out        : 0
 Number of times file system performed INPUT    : 0
 Number of times file system performed OUTPUT   : 0
 Number of IPC messages sent                    : 0
 Number of IPC messages received                : 0
 Number of signals delivered                    : 0
 Number of voluntary context switches           : 1
 Number of involuntary context switches         : 0

 #######  End of Resource Statistics  ########
```

© IBM Corporation 2006

HPM library - sample program output*(cont)*

```
Instrumented section: 1 - Label: outer call - process: 999
 file: testhpm.c, lines: 44 <--> 49
  Count: 1
  Wall Clock Time: 2.204801 seconds
  Total time in user mode: 1.00511891062802 seconds
  Exclusive duration: 1.10937 seconds

  PM_FPU_1FLOP (FPU executed one flop instruction )      :          58
  PM_CYC (Processor cycles)                              :  1664476916
  PM_MRK_FPU_FIN (Marked instruction FPU processing finished) :      0
  PM_FPU_FIN (FPU produced a result)                     :      276682
  PM_INST_CMPL (Instructions completed)                  :  1380060768
  PM_RUN_CYC (Run cycles)                                :  1664476916

  Utilization rate                          :        45.588 %
  MIPS                                      :       625.934
  Instructions per cycle                    :         0.829
  HW Float point instructions per Cycle     :         0.000
  HW floating point / user time             :         0.275 M HWflop/sec
  HW floating point rate (HW Flops / WCT)   :         0.125 M HWflops/sec
```

© IBM Corporation 2006

HPM library - sample program output*(cont)*

```
Instrumented section: 2 - Label: inner call - process: 999
 file: testhpm.c, lines: 46 <--> 48
  Count: 1
  Wall Clock Time: 1.095429 seconds
  Total time in user mode: 0.498770038043478 seconds

  PM_FPU_1FLOP (FPU executed one flop instruction )            :          45
  PM_CYC (Processor cycles)                                    :   825963183
  PM_MRK_FPU_FIN (Marked instruction FPU processing finished)  :           0
  PM_FPU_FIN (FPU produced a result)                           :      138371
  PM_INST_CMPL (Instructions completed)                        :   690029068
  PM_RUN_CYC (Run cycles)                                      :   825963183

  Utilization rate                                  :        45.532 %
  MIPS                                              :       629.917
  Instructions per cycle                            :         0.835
  HW Float point instructions per Cycle             :         0.000
  HW floating point / user time                     :         0.277 M HWflop/sec
  HW floating point rate (HW Flops / WCT)           :         0.126 M HWflops/sec
```

© IBM Corporation 2006

PM tools - hpmstat and hpmcount usage

- hpmstat -h

```
usage:
   hpmstat [-H] [-k] [-o file] [-r] [-s set] [-T] [-U] [-u] interval count
   hpmstat [-h]

where:
        interval       counting time interval (default is 1 and in seconds)
        count          number of iterations to count
        -H             count hypervisor activity only
        -h             displays this help message
        -k             count system activity only (default is to count system,
                       user and hypervisor activity)
        -o file        output file name
        -r             enable runlatch, disable counts while executing in
                       idle cycle
        -s set         pre-defined set of events (1 to 8) - see command pmlist
        -T             write time stamps instead of time in seconds
        -U             the counting time interval is microseconds
        -u             count user activity only
```

- hpmcount -h

```
usage:
   hpmcount [-a] [-H] [-k] [-o file] [-s set] command
   hpmcount [-h]

where:
        command        program to be executed
        -a             aggregate counters on POE runs
        -H             adds hypervisor activity on behalf of the process
        -h             displays this help message
        -k             adds system activity on behalf of the process
        -o file        output file name
        -s set         pre-defined set of events (1 to 8) - see command pmlist
```

© IBM Corporation 2006

hpmcount - example

- hpmcount sleep 5

```
Execution time (wall clock time): 5.02176 seconds

 ########  Resource Usage Statistics  ########

 Total amount of time in user mode            : 0.005317 seconds
 Total amount of time in system mode          : 0.002731 seconds
 Maximum resident set size                    : 140 Kbytes
 Average shared memory use in text segment    : 0 Kbytes*sec
 Average unshared memory use in data segment  : 0 Kbytes*sec
 Number of page faults without I/O activity   : 43
 Number of page faults with I/O activity      : 1
 Number of times process was swapped out      : 0
 Number of times file system performed INPUT  : 0
 Number of times file system performed OUTPUT : 0
 Number of IPC messages sent                  : 0
 Number of IPC messages received              : 0
 Number of signals delivered                  : 0
 Number of voluntary context switches         : 1
 Number of involuntary context switches       : 0

 #######  End of Resource Statistics  ########

  PM_LSU_CMPL (LSU instructions completed)          :         12321
  PM_CYC (Processor cycles)                         :        146161
  PM_INST_CMPL (Instructions completed)             :         31994
  PM_INST_DISP (Instructions dispatched)            :         34635

  Utilization rate                                :         0.008 %
  MIPS                                            :         0.006
  Instructions per cycle                          :         0.219
```

© IBM Corporation 2006

hpmcount - example*(cont)*

- hpmcount -s 8 sleep 5

```
Execution time (wall clock time): 5.009662 seconds

 ########  Resource Usage Statistics  ########

 Total amount of time in user mode              : 0.005482 seconds
 Total amount of time in system mode            : 0.001942 seconds
 Maximum resident set size                      : 140 Kbytes
 Average shared memory use in text segment      : 0 Kbytes*sec
 Average unshared memory use in data segment    : 1 Kbytes*sec
 Number of page faults without I/O activity     : 43
 Number of page faults with I/O activity        : 0
 Number of times process was swapped out        : 0
 Number of times file system performed INPUT    : 0
 Number of times file system performed OUTPUT   : 0
 Number of IPC messages sent                    : 0
 Number of IPC messages received                : 0
 Number of signals delivered                    : 0
 Number of voluntary context switches           : 1
 Number of involuntary context switches         : 0

 #######  End of Resource Statistics  ########

  PM_BR_MPRED (Branches incorrectly predicted)            :           894
  PM_BR_DISP (Instructions dispatched to the branch unit) :         10417
  PM_CYC (Processor cycles)                               :        152488
  PM_INST_CMPL (Instructions completed)                   :         31988

  Utilization rate                         :           0.008 %
  Branches mispredicted percentage         :           8.582 %
  MIPS                                     :           0.006
  Instructions per cycle                   :           0.210
```

© IBM Corporation 2006

PM tools TL5 update - counter multiplexing

- hpmcount and hpmstat support
  - -s flag now allows comma separated list of event sets to be specified
    - set "0" means all sets
  - environment variables similarly now accepts multiple comma separated sets
    - allows support in HPM library too
  - multiple groups or sets of events can now be specified via event file

© IBM Corporation 2006

hpmcount - example of multiplexing all sets

```
# hpmcount -s 0 ipc4
 Execution time (wall clock time): 64.697222 seconds

 ########  Resource Usage Statistics  ########

 Total amount of time in user mode            : 64.339401 seconds
 Total amount of time in system mode          : 0.017005 seconds
 Maximum resident set size                    : 388 Kbytes
 Average shared memory use in text segment    : 257 Kbytes*sec
 Average unshared memory use in data segment  : 24757 Kbytes*sec
 Number of page faults without I/O activity   : 140
 Number of page faults with I/O activity      : 0
 Number of times process was swapped out      : 0
 Number of times file system performed INPUT  : 0
 Number of times file system performed OUTPUT : 0
 Number of IPC messages sent                  : 0
 Number of IPC messages received              : 0
 Number of signals delivered                  : 0
 Number of voluntary context switches         : 2
 Number of involuntary context switches       : 6656

 #######  End of Resource Statistics  ########

  PM_LSU_CMPL (LSU instructions completed)                               :      7981013360
  PM_CYC (Processor cycles)                                              :     24001739529
  PM_INST_CMPL (Instructions completed)                                  :     32000866113
  PM_INST_DISP (Instructions dispatched)                                 :     31992690593
  PM_IC_MISS (Instruction cache misses)                                  :            8068
  PM_LSU_IDLE (Cycles LSU is idle)                                       :     16006473444
  PM_SNOOP (Snoop requests received)                                     :           29310
  PM_SNOOP_HIT (Snoop hits)                                              :               8
  PM_FPU_CMPL (Floating-point instructions completed (no loads or stores)) :             0
  PM_FXU_CMPL (Integer instructions completed (no loads or stores))      :     16007417946
  PM_DTLB_MISS (Data TLB misses)                                         :             674
  PM_ITLB_MISS (Instruction TLB misses)                                  :             134
  PM_BR_MPRED (Branches incorrectly predicted)                           :               0
  PM_BR_DISP (Instructions dispatched to the branch unit)                :      8004870010


  Processing time                                  :          64.005 s
  Utilization rate                                 :          98.930 %
  Instructions per cycle                           :           1.333
  MIPS                                             :         494.625 MIPS
  % Instructions dispatched that completed         :         100.026 %
  Total load and store operations                  :        7981.013 M
  Instructions per load/store                      :           4.010
  Instructions per I Cache Miss                    :           4.010
  % Cycles LSU is idle                             :          66.689 %
  Snoop hit rate                                   :           0.027 %
  HW floating point instructions per Cycle         :           0.000
  HW floating point instructions / user time       :           0.000 M HWflops/s
  HW floating point rate                           :           0.000 M HWflops/s
  Total Fixed point operations                     :       16007.418 M
  Fixed point operations per Cycle                 :           0.667
  Branches mispredicted percentage                 :           0.000 %
```

© IBM Corporation 2006

PM tools - tprof event based profiling support

- Starting with 5.3 ML 3, tprof supports event-based instruction profiling
- New options
  - -E [<event>]
    - default is PM_CYC (processor cycles)
    - other hardware events: PM_*
    - software events: EMULATION, ALIGNMENT, ISLBMISS, DSLBMISS
  - -f interval
    - 1-500 (ms) in case event is PM_CYC or one of the software events
    - 10000 to MAXINT for the other PM events
- Enhanced output
  - new configuration section
- Event profiling mode uses new trace hook (0x2FF)
  - used for samples and configuration information
  - /etc/tcfmt has new template for it

© IBM Corporation 2006

tprof - event based profiling

- Profiling setup
  - one counter is programmed to monitor selected event
  - if necessary a second counter is programmed to monitor instructions completed
  - Performance Monitoring Unit is programmed to generate an interruption when counters become negative
  - on interruption, SIA and SDA register values are captured and stored in a tracehook
  - frequency of sampling is controled by counter reload value

- Reload value calculation

```
if PM_CYC or software events are used
    init_load = 0x80000000 - (find_count_cycles(nbr_ms / tprof_cyc_mult))
else
    init_load = 0x80000000 - (nbr_events / tprof_evt_mult)

nbr_ms and nbr_events are -f arguments

find_count_cycles converts ms to number of processor cycles
```

- Three raso tunables control sampling limits
  - tprof_inst_threshold: controls minimum number of instructions between samples
  - tprof_cyc_mult: controls maximum cycles sampling frequency
  - tprof_evt_mult: controls maximum event sampling frequency

© IBM Corporation 2006

tprof - raso tunables

- tprof_inst_threshold
  - purpose: minimum number of completed instructions allowed between PM_* (including PM_CYC) event samples. If the threshold is reached 5 times consecutively before, sampling is stopped.
  - values: 1..1000..MAXINT

- tprof_cyc_mult
  - purpose: PM_CYC and software events sampling frequency multiplier
  - values: 1..1..100

- tprof_evt_mult
  - purpose: PM_* events sampling frequency multiplier
  - values: 1..1..10000

© IBM Corporation 2006

tprof - configuration report examples

- Realtime mode example

```
Configuration information
=========================
System: AIX 5.3 Node: stram Machine: 005D13DA4C00
Tprof command was:
    tprof -u -R -E PM_CYC -f 10 -r rootstring -x sleep 50
Trace command was:
    trace -a -J tprof -o rootstring.trc
Total Samples = 195
Total Elapsed Time = 1.96s
Performance Monitor based report
    Processor name: power5
    Monitored event: Processor cycles (PM_CYC)
    Sampling frequency:    10 ms
PURR was used to calculate percentages
```

- Postprocessing mode example

```
Configuration information
=========================
System: 5.3 Node: monvelo Machine: 0054BDAA4C00
Tprof command was:
    ./tprof -r toto
Tprof command used to produce input files was:
    ./tprof -c -A all -C all -r toto -x ls
Trace command was:
    trace -a -L 1000000 -T 500000 -j 000,001,002,003,38F,005,006,134,139,465,00A,234
          -o toto.trc -Call
Total Samples = 368
Total Elapsed Time = 1.84s
```

© IBM Corporation 2006

Thank You!